# On Interpolants and Variable Assignments

Pavel Jancik, Jan Kofroň
Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic
Email: name.surname@d3s.mff.cuni.cz

Simone Fulvio Rollini, Natasha Sharygina
Faculty of Informatics,
University of Lugano, Switzerland,
Email: name.surname@usi.ch

***Abstract*—Craig interpolants are widely used in program verification as a means of abstraction. In this paper, we (i) introduce *Partial Variable Assignment Interpolants (PVAIs)* as a generalization of Craig interpolants. A variable assignment focuses computed interpolants by restricting the set of clauses taken into account during interpolation. PVAIs can be for example employed in the context of DAG interpolation, in order to prevent unwanted out-of-scope variables to appear in interpolants. Furthermore, we (ii) present a way to compute PVAIs for propositional logic based on an extension of the Labeled Interpolation Systems, and (iii) analyze the strength of computed interpolants and prove the conditions under which they have the path interpolation property.**

## I. Introduction

In software model checking Craig interpolants play an important role. They are typically used to refine an abstraction of a program. Many techniques have been introduced to compute interpolants for various theories such as propositional logic, conjunctive fragments of linear arithmetic, and octagon domain. For propositional logic, McMillan's [9] and Pudlák's [11] interpolation systems are well established; they are generalized by the Labeled Interpolation Systems [6] (LISs), which permit to systematically compute interpolants of different logical strength from the same refutation.

Given two formulas $A$ and $B$ such that $A \wedge B$ is unsatisfiable, a Craig interpolant is a formula $I$ such that $A$ implies $I$, $I$ is inconsistent with $B$ and $I$ is defined over the common variables of $A$ and $B$. In other words, $I$ is an over-approximation of $A$ (which can be used to abstract the behavior of a system, represented by $A$) disjoint from $B$ (which often represents unacceptable behaviors).

In this paper, we introduce *Partial Variable Assignment Interpolants (PVAIs)* – a generalization of Craig interpolants – which, in addition to the standard subdivision of an unsatisfiable formula (the *interpolation problem*) into $A$ and $B$, is parametric in a *partial variable assignment (PVA)*. A PVA defines a *sub-problem* on which a PVAI is focused. A sub-problem is obtained from the interpolation problem by removing the clauses (constraints) satisfied by the assignment. Due to the specialization, (1) it is possible to restrict the variables occurring in an interpolant to those relevant to the sub-problem, i.e. those shared between the $A$ and $B$ parts of the sub-problem. Moreover, since the irrelevant constraints (those not occurring in the sub-problem) need not be considered by interpolation, (2) the interpolants for the sub-problem can be of smaller size, compared to Craig interpolants computed from the interpolation problem.

This work is partially supported by: ICT COST Action IC0901, the Grant Agency of the Czech Republic project 14-11384S, and Charles University Foundation grant 203-10/253297.

In the motivating example in Sec. II we show how PVAIs apply to program verification. For instance, in the context of abstract reachability graphs (ARG) (and DAG interpolation [2]), an interpolation problem is the encoding of a whole ARG (representing all paths in the ARG), while for a given ARG node $i$ the related sub-problem represents the set of paths that pass through that node. An over-approximation of the states reachable at $i$ via these paths (a *node interpolant*) can be computed by means of a PVAI. Properties of PVAIs guarantee that the interpolant contains only in-scope program variables.

An alternative approach could be to solve each sub-problem separately, which involves calling a SAT/SMT solver for each sub-problem and applying standard Craig interpolation. The method we propose allows one to perform just a single call to a solver for an interpolation problem which encompasses all the sub-problems, thus (i) processing the parts common to multiple sub-problems only once. A single solver call results in a single proof from which all the interpolants for the sub-problems are computed. The presence of a single proof, in turn, enables (ii) generating collections of interpolants which satisfy properties relevant to verification, such as path interpolation [7], [13]. Such collections are hard to obtain if multiple proofs are involved. In the case of PVAIs, a collection may consist of the interpolants associated with different sub-problems.

We also propose the new framework of *Labeled Partial Assignment Interpolation Systems* (LPAISs) – a generalization of LISs, which computes PVAIs for propositional logic. We define the notion of logical strength for LPAISs and show how introducing a partial order over LPAISs allows to systematically compare the strength of the computed interpolants (a feature intuitively relevant to verification since it affects the coarseness of the over-approximations realized by interpolants [12]). We also show how LPAISs can be used to generate collections of interpolants which enjoy the path interpolation (inductive step) property. These results can be applied in the context of ARGs, where the path interpolation property of computed node interpolants (labels) guarantees well-labeledness [10] of the ARG.

ISBN: 978-0-9835678-4-4.  

```
1: int max(int i, int j) {
2:   if (i > j)
3:     return i;
    else
4:     return j;
5: }
   // The main function
6: assert(max(random(), 0) >= 0);
```

Figure 1. Motivating example

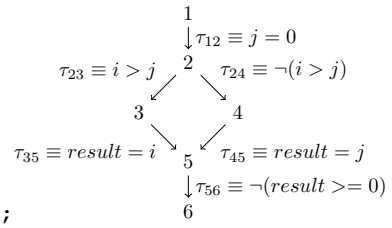


Figure 2. Abstract reachablity graph

$$\mu_1 \equiv (n_1 \Rightarrow n_2) \qquad \wedge ((n_1 \wedge n_2) \Rightarrow \tau_{12})$$
$$\mu_2 \equiv (n_2 \Rightarrow (n_3 \vee n_4)) \qquad \wedge ((n_2 \wedge n_3) \Rightarrow \tau_{23}) \wedge$$
$$\wedge ((n_2 \wedge n_4) \Rightarrow \tau_{24})$$
$$\mu_3 \equiv (n_3 \Rightarrow n_5) \qquad \wedge ((n_3 \wedge n_5) \Rightarrow \tau_{35})$$
$$\mu_4 \equiv (n_4 \Rightarrow n_5) \qquad \wedge ((n_4 \wedge n_5) \Rightarrow \tau_{45})$$
$$\mu_5 \equiv (n_5 \Rightarrow n_6) \qquad \wedge ((n_5 \wedge n_6) \Rightarrow \tau_{56})$$

$$\mathsf{Cond} \equiv n_1 \wedge \mu_1 \wedge \mu_2 \wedge \mu_3 \wedge \mu_4 \wedge \mu_5$$

Figure 3. The Cond formula

## II. Motivation

In the following, we illustrate a possible application of PVAIs, which originally motivated this work; nonetheless, the proposed PVAIs are not limited to this context. As an example, consider the source code on the left-hand side of Fig. 1 and the corresponding ARG in Fig. 2. Node $i$ is associated with location $i$ in the program. Node 1 is the initial node, while node 6 is the node representing an error location. The *edge constraints* $\tau_{ij}$ encode the semantics of the corresponding program statements. Note that $\tau_{12}$ originates from the call to the `max` function in `main`, on line 6. Further, in node 3, the parameter $i$ is the only in-scope variable; similarly in node 4 the parameter $j$ is the only in-scope variable. A variable is in-scope at a given node, if there is a path through the node where the variable is used before as well as after the node.

In the context of software verification, an important question is whether an error location is actually reachable from the initial location of a program – this is known as the *reachability problem*. The question can be answered by computing, for each node $i$, the set of states reachable at $i$ via paths in the program ARG [4], [10]. Typically, it is enough to compute an over-approximation of these states, i.e. a *node interpolant*. To this end, the ARG is converted into a Cond formula[1], which represents all execution paths in the ARG. An auxiliary *structure-encoding* Boolean variable $n_i$ is introduced for each node $i$ in the ARG; for each $i$ (except for the error node), a *node formula* $\mu_i$ is created, which encodes the labels on the outgoing edges (Fig. 3).

For illustration, we describe the meaning of $\mu_2$. The first conjunct $n_2 \Rightarrow (n_3 \vee n_4)$ expresses that after reaching node 2, a path has to proceed to a successor node (3 or 4). The second conjunct $(n_2 \wedge n_3) \Rightarrow \tau_{23}$ guarantees that if a path goes via the edge $2 \rightarrow 3$, the semantics of the edge is preserved (i.e., the constraint $\tau_{23}$ is satisfied). Similarly, the third conjunct enforces the semantics of the edge $2 \rightarrow 4$.

The Cond formula is satisfiable if and only if a feasible path exists that leads from node 1 to node 6 in the ARG. Suppose now that Cond is unsatisfiable; then a node interpolant for each node $i$ can be computed. First the ARG needs to be partitioned into $A$ and $B$ – so that $A$ corresponds to the antecedents of $i$, $B$ to all the other nodes in the ARG – and then a Craig interpolant $I$ is generated as an over-approximation of the states reachable at $i$. For instance, in the case of node 3, $A$ would be set to $n_1 \wedge \mu_1 \wedge \mu_2$ and $B$ to $\mu_3 \wedge \mu_4 \wedge \mu_5$. However, employing standard Craig interpolation in this manner to compute a node interpolant $I$ is not sufficient; out-of-scope variables might in fact belong to both $A$ and $B$, they could therefore appear in $I$, and should be consequently eliminated. Variable $j$, for example, could appear in the interpolant for node 3. Even though out-of-scope variables can be eliminated by resorting to quantification, followed by a quantifier-elimination phase, this approach is a well-known bottleneck in verification.

Computing node interpolants using PVAIs effectively solves the problem of out-of-scope program variables. Suppose that a node interpolant is to be computed for a node $k$; the created PVA assigns False to all structure-encoding variables corresponding to nodes not lying on the paths through $k$. By setting a variable $n_j$ to False, in fact, the paths via node $j$ are blocked; moreover, the whole node formula $\mu_j$ is satisfied and thus $\mu_j$ is not a part of the sub-problem for node $k$. On the other hand, the PVA assigns $n_k$ to True to express that each considered path has to pass through $k$ (the node for which the interpolant is computed). In particular, to compute an interpolant for node 3, we assign $n_3$ to True and $n_4$ to False to block the path through node 4; the rest of variables remain unassigned. This assignment satisfies (and thus removes) $n_2 \Rightarrow (n_3 \vee n_4)$, $(n_2 \wedge n_4) \Rightarrow \tau_{24}$ and $\mu_4$ from the sub-problem (see Fig. 4). In the $A$ part, the sub-problem for node 3 contains the edge labels (and consequently the program state variables) related to the path from node 1 to node 3, and in the $B$ part information related to the path from node 3 to node 6. The program state variables shared by the $A$ and $B$ parts of the sub-problem are the in-scope variables, which are exactly those that may appear in PVA interpolants.

## III. Preliminaries

A *clause* is a finite disjunction of literals. We use angle brackets $\langle\Theta\rangle$ to denote the clause built over the literals in $\Theta$. Let $\langle\Theta, p\rangle$ and $\langle\Theta', \overline{p}\rangle$ be clauses. Using variable $p$ as the *pivot*, their resolution yields the clause $\langle\Theta, \Theta'\rangle$. In the following, we consider propositional formulas in *conjunctive normal form*,

$$\pi_3 \equiv n_3 \wedge \overline{n_4}$$
$$A_3 \equiv n_1 \wedge (n_1 \Rightarrow n_2) \wedge ((n_1 \wedge n_2) \Rightarrow j = 0) \wedge \wedge ((n_2 \wedge n_3) \Rightarrow i > j)$$

$$B_3 \equiv (n_3 \Rightarrow n_5) \wedge ((n_3 \wedge n_5) \Rightarrow result = i) \wedge (n_5 \Rightarrow n_6) \wedge ((n_5 \wedge n_6) \Rightarrow \neg(result >= 0))$$

Figure 4. The $A$ and $B$ parts of the sub-problem for node 3

[1] Cond has the same meaning as ArgCond in [3].

i.e., as conjunctions (or equivalently sets) of clauses. We use $\mathsf{Var}(l)$ to denote the variable of literal $l$ and $\mathsf{Var}(A)$ for the variables occurring in the set of clauses $A$.

We adopt the definition of resolution proof from [6]: a resolution proof is a tuple $(V, E, cl, piv, s)$, where $V$ is a set of vertices in the proof, $E$ is a set of edges. Each inner vertex $v$ represents resolution of its antecedent vertex-clauses (specified by $cl$) using the pivot $piv(v)$. A refutation proof derives an empty clause in the sink vertex $s$.

Since the resolution proofs take the set of clauses as input, the input formula is first converted into a conjunction of clauses. Thus in the following we use the terms formula and set of clauses interchangeably.

A *Craig interpolant* [5] for the pair of formulas $(A, B)$ such that $A \wedge B$ is unsatisfiable is a formula $I$ such that (1) $A \Rightarrow I$, (2) $B \wedge I \Rightarrow \bot$, and (3) $\mathsf{Var}(I) \subseteq \mathsf{Var}(A) \cap \mathsf{Var}(B)$.

An *interpolant sequence* for the unsatisfiable formula $A_1 \wedge A_2 \wedge ... \wedge A_n$ is a tuple of formulas $(I_0, I_1, ....I_n)$, where $I_i$ is an interpolant for $(A_1 \wedge ... \wedge A_i, A_{i+1} \wedge ... \wedge A_n)$. If for all $i$, $I_i \wedge A_i \Rightarrow I_{i+1}$, then $(I_0, I_1, ....I_n)$ is said to satisfy the *path interpolation* (PI) property. In [7], it was proved that the path interpolation property holds for any LISs, including the well-known McMillan's and Pudlák's systems, whenever the interpolant sequence is computed from the same proof.

Let $A$ be a set of clauses. A *variable assignment* assigns either True ($\top$) or False ($\bot$) to each variable in the $\mathsf{Var}(A)$ set. The variable assignment can be seen as a conjunction of literals. A *partial variable assignment* (PVA) $\pi$ assigns values only to a subset of variables in $\mathsf{Var}(A)$. A PVA $\pi$ can be used as an assumption w.r.t. $A$ (i.e., $\pi \models A$) to restrict the set of models of $A$ to those compatible with $\pi$.

*Definition 1 (Clauses under assignment):* Let $A$ be a set of clauses and $\pi$ be a PVA over $\mathsf{Var}(A)$. We define the sets of *satisfied* clauses $A_\pi = \{\langle\Theta\rangle | \langle\Theta\rangle \in A \text{ and } \pi \models \langle\Theta\rangle\}$ and *unsatisfied* clauses $A_{\overline{\pi}} = \{\langle\Theta\rangle | \langle\Theta\rangle \in A \text{ and } \pi \not\models \langle\Theta\rangle\}$.

Satisfied clauses contain at least one literal evaluated to $\top$ under $\pi$, while, for unsatisfied clauses, every literal is either unassigned or falsified. The unsatisfied clauses $A_{\overline{\pi}}$ determine the sub-problem. We use $\pi \models l$ to express that a literal $l$ evaluates to $\top$ in a given PVA $\pi$.

## IV. Partial Variable Assignment Interpolants

In this section, we formally define *partial variable assignment interpolation*, which, in addition to the subdivision of an unsatisfiable formula into $A$ and a $B$ parts, requires specification of a PVA.

*Definition 2:* Let $R$ be an $(A, B)$-refutation and $\pi$ a partial variable assignment over $\mathsf{Var}(A \wedge B)$. A *partial variable assignment interpolant* (PVAI) is a formula $I$ such that:

(D2.1) $\pi \models A \Rightarrow I$
(D2.2) $\pi \models B \wedge I \Rightarrow \bot$
(D2.3) $\mathsf{Var}(I) \subseteq \mathsf{Var}(A_{\overline{\pi}}) \cap \mathsf{Var}(B_{\overline{\pi}})$
(D2.4) $\mathsf{Var}(I) \cap \mathsf{Var}(\pi) = \emptyset$

In the following we use $(A, B, \pi)$ to denote that a PVAI is computed from an $(A, B)$-refutation using the partial assignment $\pi$.

Since $\pi \models (A \Leftrightarrow A_{\overline{\pi}})$, D2.1 and D2.2 can be equivalently rewritten as $\pi \models A_{\overline{\pi}} \Rightarrow I$ and $\pi \models B_{\overline{\pi}} \wedge I \Rightarrow \bot$; in other words, $I$ is an interpolant for the sub-problem $(A_{\overline{\pi}} \wedge B_{\overline{\pi}})$. Note that even after removing (the satisfied) clauses, the sub-problem remains unsatisfiable (assuming $\pi$).

On the other hand, a PVAI cannot be obtained from standard interpolants by application of a partial assignment ($I[\pi]$). The reason is that, in addition to assigned variables (disallowed by D2.4), rule D2.3 excludes from the PVAI also all unassigned (out-of-scope) variables that occur in satisfied clauses only, which can still appear in $I[\pi]$.

Calling a solver multiple times can be quite resource-consuming. An $(A, B)$-refutation proof is independent of a PVA; this important fact allows to call the solver only once on the overall problem $A \wedge B$, and later to introduce various PVAs (representing relevant sub-problems) for which the PVAI can be efficiently computed.

Although Craig interpolation has many applications in program verification, verification tools often require interpolation sequences with specific properties [7]. The PVAI for all the sub-problems are computed from the same proof, thus they are related to each other. The existence of a single proof permits the application of a standard proving technique in the area of interpolation – structural induction over a refutation proof – to show various properties of PVA interpolant sequences. All the techniques where interpolants for different sub-problems are computed using different proofs (e.g., applying a solver directly on each sub-problem, or incremental solving with assumptions) do not, per se, guarantee any properties of their sequences.

## V. Labeled Partial Assignment Interpolation System

To show that PVAIs are not just a theoretical concept, we present the framework of *Labeled Partial Assignment Interpolation Systems*, a generalization of LISs [6], which computes PVAIs for propositional logic, and prove its soundness. Next, in order to prove the path interpolation property, we introduce the concept of logical strength on LPAISs, which allows one to systematically compare the strength of the generated interpolants.

In order to define LPAISs, first we have to extend the definitions of labeling functions and locality from LISs to take variable assignments into account. Note that if no variable is assigned, LPAISs are equivalent to LISs.

A labeling function assigns labels to literals in a refutation; the labeling drives the computation of an interpolant from the proof and determines its strength.

*Definition 3 (Labeling function):* Let $L = (S, \sqsubseteq, \sqcap, \sqcup)$ be the lattice of Fig. 6, where $S = \{\bot, a, b, ab, d^+\}$ and $\bot$ is the least element, and let $R = (V, E, cl, piv, s)$ be a resolution proof over a set of literals Lit. A function $\mathsf{Lab}_{R,L} : V \times \mathsf{Lit} \to S$

| | | | |
|---|---|---|---|
| Leaf $v$: | | $\langle\Theta\rangle, [I]$ | |
| $I =$ | $\langle\Theta\rangle[\pi]\rVert_{b,v,\mathsf{Lab}}$ | if $\langle\Theta\rangle \in A_{\overline{\pi}}$ | Hyp-$A_{\overline{\pi}}$ |
| | $\neg\langle\Theta\rangle[\pi]\rVert_{a,v,\mathsf{Lab}}$ | if $\langle\Theta\rangle \in B_{\overline{\pi}}$ | Hyp-$B_{\overline{\pi}}$ |
| | $\top$ | if $\langle\Theta\rangle \in A_\pi \cup B_\pi$ | Hyp-$A_\pi$, Hyp-$B_\pi$ |
| Inner vertex $v$: | | $\dfrac{v_1 : \langle p, \Theta_1\rangle, [I_1] \qquad v_2 : \langle \overline{p}, \Theta_2\rangle, [I_2]}{\langle\Theta_1, \Theta_2\rangle, [I]}$ | |
| $I =$ | $I_1 \vee I_2$ | if $\mathsf{Lab}(v_1, p) \sqcup \mathsf{Lab}(v_2, \overline{p}) = a$ | Res-$a$ |
| | $I_1 \wedge I_2$ | if $\mathsf{Lab}(v_1, p) \sqcup \mathsf{Lab}(v_2, \overline{p}) = b$ | Res-$b$ |
| | $(I_1 \vee p) \wedge (I_2 \vee \overline{p})$ | if $\mathsf{Lab}(v_1, p) \sqcup \mathsf{Lab}(v_2, \overline{p}) = ab$ | Res-$ab$ |
| | $I_2$ | if $\mathsf{Lab}(v_1, p) = d^+$ | Res-$d^+$ |
| | $I_1$ | if $\mathsf{Lab}(v_2, \overline{p}) = d^+$ | Res-$d^+$ |

Figure 5. Labeled Partial Assignment Interpolation System

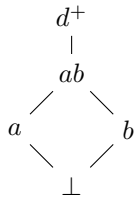


Figure 6. Lattice of labels (according to $\sqsubseteq$)

is called *labeling function* for a refutation $R$ iff $\forall v \in V$ and $\forall l \in \mathsf{Lit}$, $\mathsf{Lab}_{R,L}$ satisfies the following conditions:

(D3.1) $\mathsf{Lab}_{R,L}(v, l) = \bot$ if and only if $l \notin cl(v)$, and
(D3.2) $\mathsf{Lab}_{R,L}(v, l) = \mathsf{Lab}_{R,L}(v_1, l) \sqcup \mathsf{Lab}_{R,L}(v_2, l)$, where $v_1$, $v_2$ are the predecessor vertices.

From condition D3.2 it follows that the labeling function is fully determined once the labels in the leaves have been specified. We omit subscripts $R$ and $L$ if clear from the context.

*Naming conventions:* Let us assume a pair of sets of clauses $(A, B)$ and a PVA $\pi$. The clause sets are split into four groups, the unsatisfied clauses $A_{\overline{\pi}}$ and $B_{\overline{\pi}}$ which specify the sub-problem and are taken into account during interpolation, and the satisfied clauses $A_\pi$ and $B_\pi$, which are disregarded.

We distinguish among the following kinds of variables, depending on the standard notions of locality and sharedness, as well as on where the variables appear in the four groups of clauses. We say that a variable $k$ is *unassigned* if $k \notin \mathsf{Var}(\pi)$. An unassigned variable $k$ is:

| | |
|---|---|
| $A_{\overline{\pi}}$-*local* | if $k \in \mathsf{Var}(A_{\overline{\pi}})$ and $k \notin \mathsf{Var}(B_{\overline{\pi}})$ |
| $B_{\overline{\pi}}$-*local* | if $k \notin \mathsf{Var}(A_{\overline{\pi}})$ and $k \in \mathsf{Var}(B_{\overline{\pi}})$ |
| $A_{\overline{\pi}}B_{\overline{\pi}}$-*shared* | if $k \in \mathsf{Var}(A_{\overline{\pi}})$ and $k \in \mathsf{Var}(B_{\overline{\pi}})$ |
| $A_{\overline{\pi}}B_{\overline{\pi}}$-*clean* | if $k \notin \mathsf{Var}(A_{\overline{\pi}})$ and $k \notin \mathsf{Var}(B_{\overline{\pi}})$ |

The properties above are independent of the occurrence of $k$ in $\mathsf{Var}(A_\pi)$ and $\mathsf{Var}(B_\pi)$. The "clean" variables occur only in the satisfied clauses, thus are out-of-scope and cannot appear in a PVA interpolant.

We say that a variable $k$ is *McMillan-labeled* if, whenever $k$ is $A_{\overline{\pi}}B_{\overline{\pi}}$-shared or $A_{\overline{\pi}}B_{\overline{\pi}}$-clean, $k$ is labeled $b$ (the labels of the remaining variables are not limited to $b$). If all variables are McMillan-labeled, a LIS reduces to McMillan's interpolation system [6], which yields the strongest interpolant that LISs (and LPAISs) can produce from a given refutation proof.

A variable $k$ is labeled *consistently* if all occurrences of $k$ in a refutation have the same label.

Not all labeling functions can be used to generate interpolants; in LPAIS, interpolants are computed if a locality preserving labeling is used.

*Definition 4:* A labeling function $\mathsf{Lab}$ for an $(A, B, \pi)$-refutation $R$ is *locality preserving* iff $\forall v \in V, \forall l \in cl(v)$:

(D4.1) $\mathsf{Lab}(v, l) = d^+ \Leftrightarrow \pi \models l$
(D4.2) $\mathsf{Var}(l)$ is unassigned and $A_{\overline{\pi}}$-local $\Rightarrow \mathsf{Lab}(v, l) = a$
(D4.3) $\mathsf{Var}(l)$ is unassigned and $B_{\overline{\pi}}$-local $\Rightarrow \mathsf{Lab}(v, l) = b$
(D4.4) $\mathsf{Var}(l)$ is unassigned and $A_{\overline{\pi}}B_{\overline{\pi}}$-clean $\Rightarrow$ it is consistently labeled $a$ or $b$.

Locality constraints provide freedom in labeling $A_{\overline{\pi}}B_{\overline{\pi}}$-shared and $A_{\overline{\pi}}B_{\overline{\pi}}$-clean variables; the choice of labels directly affects the strength of the computed interpolants. The label of $A_{\overline{\pi}}B_{\overline{\pi}}$-shared variables can be set freely to $a$, $b$, or $ab$. The same holds for falsified literals; their labels are irrelevant since they are removed by the assignment filter (defined below).

The D4.2 and D4.3 rules are equivalent to the locality requirements of LIS, where $A$-local and $B$-local variables must be labeled $a$ and $b$, respectively. D4.1 concerns the satisfied literals. The label $d^+$ is used in the interpolation process to identify resolutions with an assigned pivot and parts of the proof which are not relevant to the sub-problem. The D4.4 requirement is specific to PVAI and deals with variables which occur in the satisfied clauses only. The requirement guarantees that such variables do not occur in the interpolant, because $ab$-resolution cannot be applied. Further, note that for the empty assignment the locality constraints reduce to those of LISs, since D4.1 and D4.4 do not apply to any literal.

*Filters:* For a clause $\langle\Theta\rangle$, a labeling function $\mathsf{Lab}$, a resolution-proof vertex $v \in V$, and a label $c$, we define the *match filter* $\rVert$ as $\langle\Theta\rangle\rVert_{c,v,\mathsf{Lab}} = \{l \in \langle\Theta\rangle \mid c = \mathsf{Lab}(v, l)\}$; it preserves only the literals with the specified label. Similarly, we define the *upward filter* $\restriction$ as $\langle\Theta\rangle\restriction_{c,v,\mathsf{Lab}} = \{l \in \langle\Theta\rangle \mid c \sqsubseteq \mathsf{Lab}(v, l)\}$; it preserves the literals with labels above $c$ in Fig. 6. The subscripts $\mathsf{Lab}, v$ are omitted if clear from the context. Given a partial assignment $\pi$ and a clause $\langle\Theta\rangle$, we also define the *assignment filter* $\langle\Theta\rangle[\pi] = \{l \in \langle\Theta\rangle \mid \mathsf{Var}(l) \notin \mathsf{Var}(\pi))\}$, which removes all the assigned literals (satisfied and falsified ones).

Moreover, we assume that filters have a higher precedence than negation. E.g., $\neg\langle\Theta\rangle[\pi]\restriction_a$ can be equivalently rewritten as $\neg(\langle\Theta\rangle[\pi]\rVert_a)$.

An interpolation system is a procedure for computing an interpolant from a refutation. It assigns a partial *vertex-*

*interpolant* to each vertex of the refutation, yielding the final interpolant at the sink vertex.

*Definition 5:* For a locality preserving labeling function $\mathsf{Lab}$ and an ($A$, $B$, $\pi$)-refutation $R$, Fig. 5 defines the *Labeled Partial Assignment Interpolation System* $\mathsf{Lpaltp}(\mathsf{Lab}, R)$.

An LPAIS produces interpolants in the following way: first the vertex-interpolants for leaves of the refutation proof are computed using the rules in the upper part of Fig. 5 (hypothesis rules). Depending on the occurrence of the vertex-clause $\langle\Theta\rangle$ in $A$ or $B$ sets, the corresponding rule describes the transformation of the vertex-clause into a vertex-interpolant. Later, going down through the proof from leaves to the sink, the vertex-interpolants for inner vertices are computed using rules in the lower part of Fig. 5. The labels assigned to the pivots determine how vertex-interpolants of both predecessors are combined. This process ends at the sink vertex where the PVAI is derived. The interpolants are computed in time linear to the size of the proof.

The main difference compared to LISs are the additional $d^+$ rules. For instance, consider the last rule, where $\mathsf{Lab}(v_2, \overline{p}) = d^+$. In contrast to the standard rules, the partial interpolant is simpler, because it does not contain $I_2$, omitted due to the variable assignment. Generally, these rules *cut out* the satisfied sub-tree of the proof. Usually, the later in the refutation the assigned variable is resolved, the larger sub-tree is pruned and the smaller the resulting interpolant is.

The differences between LPAISs and LISs are motivated by the way variable assignments work. The new $d^+$ rules can be seen as a specialization of the $ab$ resolution rule if a PVA $\pi$ is assumed. A similar relationship holds for the hypothesis rules in the leaves of a refutation. These rules are equivalent to LIS hypothesis rules if applied on a clause under the assumed assignment. The changes we introduce w.r.t. LISs are of two kinds: those in LPAISs rules force specialization of the interpolant on a sub-problem, while the changes in the locality constraints remove unassigned out-of-scope variables from the interpolant.

*Theorem 1 (Correctness):* $\mathsf{Lpaltp}(\mathsf{Lab}, R)$, for an ($A$, $B$, $\pi$)-refutation $R$ and a locality preserving labeling function $\mathsf{Lab}$, generates a partial variable assignment interpolant.

*Proof sketch:* By structural induction over $R$ we show that, for each vertex $v$ of a resolution proof, the following invariants hold:

$$\pi \models A \wedge \neg\langle\Theta\rangle\restriction_{a,v,\mathsf{Lab}} \Rightarrow I_v$$
$$\pi \models B \wedge \neg\langle\Theta\rangle\restriction_{b,v,\mathsf{Lab}} \Rightarrow \neg I_v$$

$I_v$ is the partial vertex-interpolant and $\langle\Theta\rangle$ is a vertex-clause of $v$. These invariants yield the PVAI constraints (D2.1, D2.2) at the sink vertex, where $\neg\langle\Theta\rangle = \top$. The full proof can be found in [8]. □

The attentive reader may notice that the locality constraints, as well as the way LPAISs compute interpolants, are *symmetric* for the $A_\pi$ and $B_\pi$ sets of satisfied clauses. It reflects the fact that these clauses are not a part of the sub-problem under consideration, thus irrelevant for PVAI interpolants. Given a fixed $\pi$, the satisfied clauses can be moved freely between the $A$ and $B$ sets; both computed interpolants and locality of the labeling functions are not affected if satisfied clauses are moved. This fact allows us to articulate the *strength* theorem in an elegant way.

### A. Strength

Interpolation systems based on labeling provide some freedom in the choice of labels (e.g., for shared variables); this choice affects the resulting interpolants, in particular their strength. In the following we investigate this relationship in more detail.

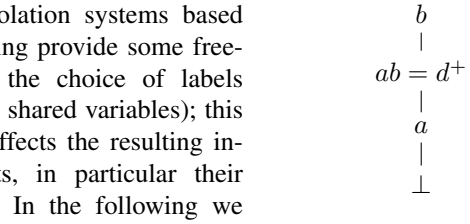


Figure 7. Strength ordering ($\preceq$)

*Definition 6 (Strength order):* Let $\preceq$ be a pre-order relation defined on the set of labels $S = \{\bot, a, b, ab,\ d^+\}$ as: $b \preceq ab = d^+ \preceq a \preceq \bot$ (see Fig. 7). Let $\mathsf{Lab}$ and $\mathsf{Lab}'$ be labeling functions for a refutation $R$. We say $\mathsf{Lab}$ *is stronger than* $\mathsf{Lab}'$, denoted as $\mathsf{Lab} \preceq \mathsf{Lab}'$, if for all vertices $v \in V$ and for all literals $l \in cl(v)$ it holds that $\mathsf{Lab}(v, l) \preceq \mathsf{Lab}'(v, l)$.

Note that labels $ab$ and $d^+$ are of the same strength and can be exchanged if the locality requirements permit; $b$ is the strongest label, while $a$ is the weakest one a literal can get.

The following theorem states that the introduced strength order on labeling functions also orders the produced interpolants by logical strength.

*Theorem 2 (Interpolant strength):* Let $\mathsf{Lab}$ be a locality preserving labeling function for an ($A$, $B$, $\pi$)-refutation $R$, and $\mathsf{Lab}'$ be a locality preserving labeling function for ($A$, $B$, $\pi'$)-$R$. Let $I$ be a partial variable assignment interpolant for $\mathsf{Lpaltp}(\mathsf{Lab}, R)$ and $I'$ be a PVAI for $\mathsf{Lpaltp}(\mathsf{Lab}', R)$.

If $\mathsf{Lab} \preceq \mathsf{Lab}'$ then $\pi, \pi' \models I \Rightarrow I'$.

Note that when $\pi$ and $\pi'$ are *empty* assignments, we obtain exactly the theorem on interpolant strength from [6]. Also note that the theorem permits different variable assignments for the interpolants. Thus it relates the interpolants generated for different sub-problems (e.g., interpolants considering different sets of paths through a given ARG node). Since both $\pi$ and $\pi'$ are assumptions of the formula $I \Rightarrow I'$, the theorem applies to cases common to both sub-problems (i.e., to the shared paths). Both interpolants ($I$ and $I'$) have to be computed using the same $A$ and $B$ parts, thus interpolants for different ARG nodes cannot be compared using this theorem; a generalization in this direction is shown in the following sub-section.

In the following proof, we need a new type of filter. Let $\mathsf{Lab}$ and $\mathsf{Lab}'$ be labeling functions to be compared by strength and $v$ be a vertex of the refutation proof. The new *weakened-labels filter* $\Vert_v^{\mathsf{Lab},\mathsf{Lab}'}$ preserves the literals whose label is weaker in $\mathsf{Lab}'$ than in $\mathsf{Lab}$. E.g., the filter preserves a literal $l$ if the strongest labels $b$ ($\mathsf{Lab}(v, l) = b$) is weakened into label $a$ or

$ab$ in $\mathsf{Lab}'(v, l)$, while it filters-out a literal if both functions assign label $a$ to it. The vertex and the labeling functions are omitted if clear from the context.

*Proof sketch (Theorem 2):* By structural induction over $R$, we show that for each vertex of the resolution proof the following invariant holds:

$$\pi, \pi' \models I_v \wedge \neg\langle\Theta\rangle|_v \Rightarrow I'_v$$

$\langle\Theta\rangle$ is the vertex-clause, $I_v$ and $I'_v$ are the partial vertex-interpolants for the vertex $v$ as generated by our interpolation system using the labeling functions $\mathsf{Lab}$ and $\mathsf{Lab}'$, respectively. The full proof in [8] shows that the invariant holds for all combinations of rules that can be used to define the vertex-interpolants $I_v$ and $I'_v$. □

Similarly to LISs, for a fixed variable assignment there is a lattice of LPAISs ordered according to the strength of labeling functions. The top element of the lattice involves the strongest labeling function, which assigns label $b$ to $A_{\overline{\pi}}B_{\overline{\pi}}$-shared and $A_{\overline{\pi}}B_{\overline{\pi}}$-clean variables, while the labeling function of the bottom element assigns label $a$ to them. Theorem 2 claims that LPAISs produce interpolants ordered by strength according to the lattice.

### B. Path interpolation property

Several verification approaches such as [3], [10], [14] depend on the *path interpolation* property (PI). In [13] the authors show that LISs can be employed to generate path interpolants by providing a sequence of labeling functions that are decreasing in terms of strength. In this subsection we study conditions for labeling functions that have to be satisfied in order to guarantee the PI property of interpolant sequences generated by LPAISs.

First, we show that the PI property holds if the same partial assignment along a sequence is used to compute the interpolants (i.e., considering the same set of paths at different ARG nodes). Later on, we generalize the result to permit different partial assignments for particular interpolants (i.e., relating node interpolants).

*Fixed PVA:* To show the PI property, it is enough to prove that, for any consecutive interpolants in the sequence, it holds: $I \wedge S \Rightarrow I'$, where $I$ is an interpolant for $(A, S \cup B, \pi)$, $I'$ is an interpolant for $(A \cup S, B, \pi)$, and $S$ is a set of clauses.

For LISs, [13] defines a set of *labeling constraints* on the labeling functions used to compute the interpolants $I$ and $I'$; if the labeling constraints are satisfied, the interpolants have the PI property. However, we prove the PI property in another way, more suitable for LPAISs. Given a labeling function to compute the interpolant $I$, we define the strongest labeling function which can be used to compute the successor interpolant $I'$.

*Definition 7:* Let $\mathsf{Lab}$ be a labeling function for an $(A, S \cup B, \pi)$-refutation $R$. The *strongest successor labeling* function $\mathsf{Lab}^S$ (for the set $S$) is defined in Fig. 8.

It is easy to see that $\mathsf{Lab}^S$ is a valid labeling function and that if $\mathsf{Lab}$ is locality preserving, then $\mathsf{Lab}^S$ is locality preserving for $(A \cup S, B, \pi)$. Hence, $\mathsf{Lab}^S$ can be used to compute an interpolant for $(A \cup S, B, \pi)$.

The first alternative (D7.1) forces label $a$ for all literals which become $(A_{\overline{\pi}} \cup S_{\overline{\pi}})$-local due to the shift of the clauses in $S$ from the $B$ to the $A$ part. Any locality preserving function $\mathsf{Lab}'$ has to also assign the label $a$ to these literals. So, it is easy to see that if $\mathsf{Lab} \preceq \mathsf{Lab}'$ then also $\mathsf{Lab}^S \preceq \mathsf{Lab}'$. This expresses the meaning of *strongest*. Moreover, $\mathsf{Lab} \preceq \mathsf{Lab}^S$, because either the labels are equal or the weakest label $a$ is used in the labeling $\mathsf{Lab}^S$.

The following lemma states the PI property for the strongest successor labeling.

*Lemma 1:* Let $\mathsf{Lab}$ be a locality preserving labeling function for an $(A, S \cup B, \pi)$-refutation $R$ and let $\mathsf{Lpaltp}(\mathsf{Lab}, R) = I$. Let $\mathsf{Lab}^S$ be the strongest successor labeling for $\mathsf{Lab}$ and $S$, and $\mathsf{Lpaltp}(\mathsf{Lab}^S, (A \cup S, B, \pi)) = I'$.
Then $\pi \models I \wedge S \Rightarrow I'$.

*Proof sketch:* By structural induction over $R$, we show that for each vertex $v$ of the resolution proof the following invariant holds:

$$\pi \models I_v \wedge S \wedge \neg\langle\Theta\rangle|_v \Rightarrow I'_v$$

$\langle\Theta\rangle$ is the vertex-clause, $I_v$ and $I'_v$ are the partial vertex-interpolants for the vertex $v$ as generated by our interpolation system using the labeling functions $\mathsf{Lab}$ and $\mathsf{Lab}^S$, respectively. The full proof can be found in [8]. □

Lemma 1 guarantees the PI property only if the sequence of the strongest successors labeling functions is used. Below we generalize this result in such a way that the strength of the labeling function can decrease along the sequence; Theorem 3 states the main result for a fixed partial assignment – the path interpolation property.

*Theorem 3:* Let $\mathsf{Lab}$ and $\mathsf{Lab}'$ be locality preserving labeling functions for an $(A,\ S \cup B,\ \pi)$-refutation $R$ and $(A \cup S,\ B,\ \pi)$-$R$, respectively. Let $\mathsf{Lpaltp}(\mathsf{Lab}, R) = I$ and $\mathsf{Lpaltp}(\mathsf{Lab}', R) = I'$.
If $\mathsf{Lab} \preceq \mathsf{Lab}'$ then $\pi \models I \wedge S \Rightarrow I'$.

*Proof:* Let $I^S$ be the partial variable interpolant for the strongest successor labeling function $\mathsf{Lab}^S$. From Lemma 1 it holds that $\pi \models I \wedge S \Rightarrow I^S$. As shown above $\mathsf{Lab}^S \preceq \mathsf{Lab}'$; so Theorem 2 can be applied and $\pi \models I^S \Rightarrow I'$. □

The result in this case is the same as for LISs. In the following we focus on the case when different PVAs are used, and the situation becomes more challenging.

*Different PVAs:* The goal to prove when different partial assignments $\pi$ and $\pi'$ are used to compute interpolants $I$ and $I'$ (respectively) is:

$$\pi, \pi' \models I \wedge S \Rightarrow I'$$

Looking back at the motivating example, for each node in the ARG a different partial variable assignment is typically used; thus, the generalization done in this section is needed to relate the interpolants of adjacent ARG nodes. Assume node

$$\mathsf{Lab}^S(v,l) = \begin{cases} a & \text{if } \mathsf{Var}(l) \in \mathsf{Var}(S_{\overline{\pi}}) \ \wedge \ \mathsf{Var}(l) \notin \mathsf{Var}(B_{\overline{\pi}}) \ \wedge \ \mathsf{Var}(l) \notin \mathsf{Var}(\pi) \quad (D7.1) \\ \mathsf{Lab}(v,l) & \text{otherwise} \quad (D7.2) \end{cases}$$

Figure 8. Strongest successor labeling function

interpolants $I_2$ for node 2 and $I_3$ for node 3. The desired property is then $I_2 \wedge \tau_{23} \Rightarrow I_3$ (well-labeledness in the context of ARGs [3], [10]), which follows from the aforementioned goal. In Theorem 4, we work out the conditions the labeling functions (for $I_2$ and $I_3$) have to satisfy so that the interpolants have the desired property.

*Assignments:* Having two different PVAs $\pi$ and $\pi'$, the expression $(\pi, \pi')$ represents the PVA formed by the union of $\pi$ and $\pi'$. We say that a PVA $\sigma$ is an *extension* of a PVA $\pi$, if $\sigma \Rightarrow \pi$ (viewing the PVAs as conjunctions of literals). In other words, $\sigma$ can be created from $\pi$ by assigning additional variables. In case of conflicting $\pi$ and $\pi'$ (assigning one $\top$ and the other $\bot$ to a particular variable), the goal above holds trivially and therefore we omit the case from now on.

*Definition 8:* We say that the variable is *assignable* if it is McMillan-labeled and not $A_{\overline{\pi}}$-local.

Each assignable variable must have label $b$, therefore, after assigning it, its label becomes weaker. The following theorem states the main result for different PVAs.

*Theorem 4:* Let $\mathsf{Lab}$ be a locality preserving labeling function for an $(A, S \cup B, \pi)$-refutation $R$ and let $I = \mathsf{Lpaltp}(\mathsf{Lab}, (A, S \cup B, \pi))$. Let $\mathsf{Lab}'$ be a locality preserving labeling function for $(A \cup S, B, \pi')$-$R$ and let $I' = \mathsf{Lpaltp}(\mathsf{Lab}', (A \cup S, B, \pi'))$.

Suppose that (i) $A_{\overline{\pi}} \subseteq A_{\overline{\pi}'}$, (ii) $B_{\overline{\pi}'} \subseteq B_{\overline{\pi}}$, (iii) the variables assigned by $\pi'$ and not by $\pi$ are assignable in $\mathsf{Lab}$, and (iv) the variables assigned by $\pi$ and not by $\pi'$ are not $B_{\overline{\pi}'}$-local.

If $\mathsf{Lab} \preceq \mathsf{Lab}'$ then it holds $\pi, \pi' \models I \wedge S \Rightarrow I'$.

Intuitively, the constraints (i) and (ii) prevent from comparing interpolants of unrelated sub-problems. The only way to violate the constraint (i) $A_{\overline{\pi}} \subseteq A_{\overline{\pi}'}$ is to assign a new variable by $\pi'$. In terms of ARGs, it means that $\pi'$ blocks some paths in addition to those blocked by $\pi$. The interpolant $I$ over-approximates the states reachable in the corresponding node via non-blocked paths in the $A$ part. If the assignment $\pi'$ blocks some paths related to $I'$ in addition to those blocked by $\pi$, then $I'$ may not cover (over-approximate) the states coming from the blocked paths, thus it may be not implied by $I$. A similar reasoning can be used for (ii).

*Proof sketch:* The overall idea of the proof is shown in Fig. 9. The proof consists of four simpler steps. In the first step (① → ②) new variables get assigned by $\pi'$, in the second step (② → ③) the clauses of $S$ are moved. In the third step (③ → ④) the assignment $\pi$ is removed, in the last step (④ → ⑤) the labeling function is weakened. In the second line of Fig. 9, it is expressed how the interpolation problem is divided into $A$ and $B$ parts and which PVA is used. In all but the second step the division into $A$ and $B$ parts does not change, thus Theorem 2 can be used to relate particular interpolants with each other via implications; in the second step the partial variable assignment does not change, so Theorem 3 is utilized.

To be able to apply this scheme (Theorems 2 and 3), locality preserving labeling functions of decreasing strength are needed. The third line of Fig. 9 specifies a labeling function for each step. The idea of the approach is similar to the one used for fixed variable assignments. In each step, we create the strongest possible labeling function; in particular for the first step (① → ②) we create an *extended-assignment labeling* function ($\mathsf{Lab}^+_{\pi \to (\pi,\pi')}$) – the strongest locality-preserving labeling function if new variables get assigned. For the second step (② → ③) we use the strongest successor labeling function as defined in Def. 7. For the third step (③ → ④) we create a *restricted-assignment labeling* function ($\mathsf{Lab}^-_{(\pi,\pi') \to \pi'}$) – the strongest locality-preserving labeling function if variables get unassigned. For the sake of space, we skip the definitions of the aforementioned labeling functions and proofs of the required properties; they can be found in [8].

Via the above construction we create the strongest locality-preserving labeling function ($\mathsf{Lab}^-_{(\pi,\pi') \to \pi'}$) for $(A \cup S, B, \pi')$ which satisfies $\mathsf{Lab} \preceq \mathsf{Lab}^-_{(\pi,\pi') \to \pi'}$. In the last step (④ → ⑤) we decrease the strength into $\mathsf{Lab}'$, in the same way as it is done for $\mathsf{Lab}^S$ in Theorem 3.

The last line of Fig. 9 shows how the interpolants in each step are related to each other and how the overall claim of this theorem follows from the particular steps. □

### C. Application to ARGs

While the locality constraints are simple to satisfy for a single interpolant, the situation becomes more complicated if several interpolants need to be related by the path interpolation property. In such a case, the labels of the literals have to be chosen in an appropriate way. In the following, we briefly discuss how to set labels for ARG nodes (using the same encoding as in our motivating example) to apply Theorem 4 and, thus, to obtain well-labeled node interpolants.

Recall that in ARGs there are two kinds of variables – (1) *structure encoding* ($n_i$), which can be assigned, and (2) program variables, which are not assigned. The first rule is that the structure encoding variables have to be McMillan-labeled (obtaining the strongest possible labels). This rule and the properties of ARG encoding are enough to satisfy the (i)–(iv) requirements of Theorem 4.

Only the last requirement – $\mathsf{Lab}_i \preceq \mathsf{Lab}_j$ – restricts also the labels for program variables. It is easily satisfied in ARGs by a quite simple general rule: once an $A_{\overline{\pi}}B_{\overline{\pi}}$-shared or an $A_{\overline{\pi}}B_{\overline{\pi}}$-clean literal gets a label weaker than the strongest label $b$ at a node, the same or a weaker label has to be assigned at all its successor nodes, until it becomes $A_{\overline{\pi}}$-local.

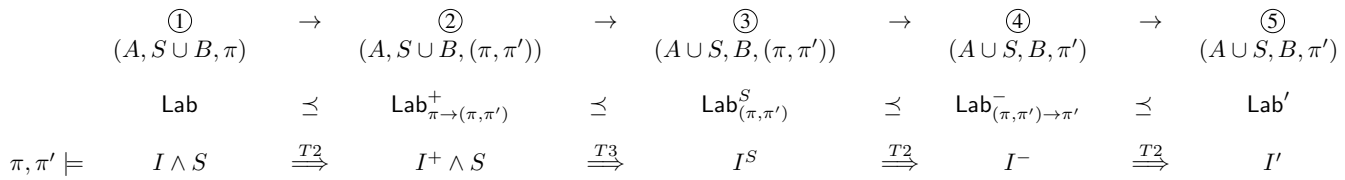


Figure 9. Idea of Theorem 4

Apparently, if for all nodes in an ARG the strongest possible labeling functions are used (i.e., all variables are McMillan-labeled), the aforementioned rules on labeling functions are satisfied, and well-labeled node interpolants are obtained.

A well-known inherent property of node interpolants is that for a path $p$ in ARG the resulting node interpolants do not form path interpolants. A node interpolant summarizes information about all paths via the node. To be able to express this "summary", the variables shared (between $A$ and $B$) on any path via the node need to be employed; we call these in-scope variables. However these variables are not necessarily $AB$-shared in the selected path $p$.

Still, path interpolants for a single path can be computed from the overall problem by means of PVAIs. Using a PVA that blocks all paths except for the one of interest, LPAISs yield path interpolants focused only on that path and over the variables shared on that path.

## VI. Related work

To the best of our knowledge, the only strongly related works in this area are [1], [3].

The approach of [3], implemented in the UFO tool, can handle linear integer arithmetic. The main idea of the technique is to linearize a DAG into a single path; after that, standard path interpolants are computed and, if out-out-scope variables are present in the interpolants, quantification is used to remove these variables. So, in general the approach leads to quantified interpolants, while LPAISs yield quantifier-free interpolants.

In [1], the authors present a different solution to the problem of out-of-scope variables. Instead of quantification, the following operations are proposed to remove them: (a) assigning constants to variables in the interpolant ($\top$ or $\bot$ in case of propositional logic) or (b) modifying the structure of the DAG encoding. Comparing to (a), our approach is more general. We naturally handle any provided assignments, thus it is possible to assign additional variables to obtain the same interpolant as suggested by [1]. Moreover, we provide more flexibility, e.g., in the case of $A_{\overline{\pi}}B_{\overline{\pi}}$-clean variables one may choose either label $b$ to obtain a stronger interpolant, or label $a$ to get a weaker one. In our work we also show the constraints under which a property relevant to verification – the path interpolation property – holds, which is not guaranteed in [1].

An aspect common to the above approaches is that they are applied as post-processing techniques, after an interpolant has been computed and only if it contains out-of-scope variables. On the contrary, our method is integrated into the computation of the interpolant, and simplifies the proof on the fly according to the corresponding variable assignment, yielding a possibly smaller interpolant.

## VII. Conclusion

In this paper, we introduced the new concept of Partial Variable Assignment Interpolants, which, unlike Craig interpolants, permits specialization to sub-problems specified in the form of variable assignments. We showed how PVAIs find application in the context of Abstract Reachability Graphs and DAG interpolation. We also developed the new framework of Labeled Partial Assignment Interpolation Systems, which can be used to compute PVAIs for propositional logic, and showed its properties.

As future work, we plan to extend the framework of LPAISs and to introduce a PVA interpolation system for linear integer arithmetic – a theory particularly relevant to program verification.

*Acknowledgment.:* Special thanks go to Ondřej Šerý for his valuable contribution.

## References

[1] Albarghouthi, A., Gurfinkel, A.: DAG-Interpolation for Software Model Checking (2013), http://cav2013.forsyte.at/files/aws_albarghouthi.pdf
[2] Albarghouthi, A., Gurfinkel, A., Chechik, M.: Craig Interpretation. In: SAS '12. LNCS, vol. 7460, pp. 300–316 (2012)
[3] Albarghouthi, A., Gurfinkel, A., Chechik, M.: From Under-Approximations to Over-Approximations and Back. In: TACAS '12. LNCS, vol. 7214, pp. 157–172 (2012)
[4] Albarghouthi, A., Li, Y., Gurfinkel, A., Chechik, M.: Ufo: A Framework for Abstraction- and Interpolation-Based Software Verification. In: CAV '12. LNCS, vol. 7358, pp. 672–678 (2012)
[5] Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. of Symbolic Logic pp. 269–285 (1957)
[6] D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: VMCAI'10. LNCS, vol. 5944, pp. 129–145 (2010)
[7] Gurfinkel, A., Rollini, S.F., Sharygina, N.: Interpolation Properties and SAT-Based Model Checking. In: ATVA '13. LNCS, vol. 8172, pp. 255–271 (2013)
[8] Jančík, P., Kofroň, J.: On Partial Variable Assignment Interpolants. Tech. Rep. 2013/5, Dept. of Distributed and Dependable Systems, Charles University in Prague (2013), http://d3s.mff.cuni.cz/publications/download/D3S-TR-2013-05-PVAI.pdf
[9] McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: CAV '03. LNCS, vol. 2725, pp. 1–13 (2003)
[10] McMillan, K.L.: Lazy Abstraction with Interpolants. In: CAV '06. LNCS, vol. 4144, pp. 123–136 (2006)
[11] Pudlák, P.: Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. Journal of Symbolic Logic 62(3), 981–998 (1997)
[12] Rollini, S., Alt, L., Fedyukovich, G., Hyvärinen, A., Sharygina, N.: PeRIPLO: A Framework for Producing Effective Interpolants in SAT-Based Software Verification. In: LPAR (2013)
[13] Rollini, S.F., Sery, O., Sharygina, N.: Leveraging Interpolant Strength in Model Checking. In: CAV '12. LNCS, vol. 7358, pp. 193–209 (2012)
[14] Vizel, Y., Grumberg, O.: Interpolation-sequence based Model Checking. In: FMCAD '09. pp. 1–8. IEEE (2009)